

Ein Benchmark zum Formulieren der Erwartungshaltung an Codequalität

---

Und bei dir so?

**CQSE**

Nils Göde



Software-**Audits**



Kontinuierliche **Qualitäts-**  
und **Testkontrolle**



 **Teamscale**

**CQSE** GmbH



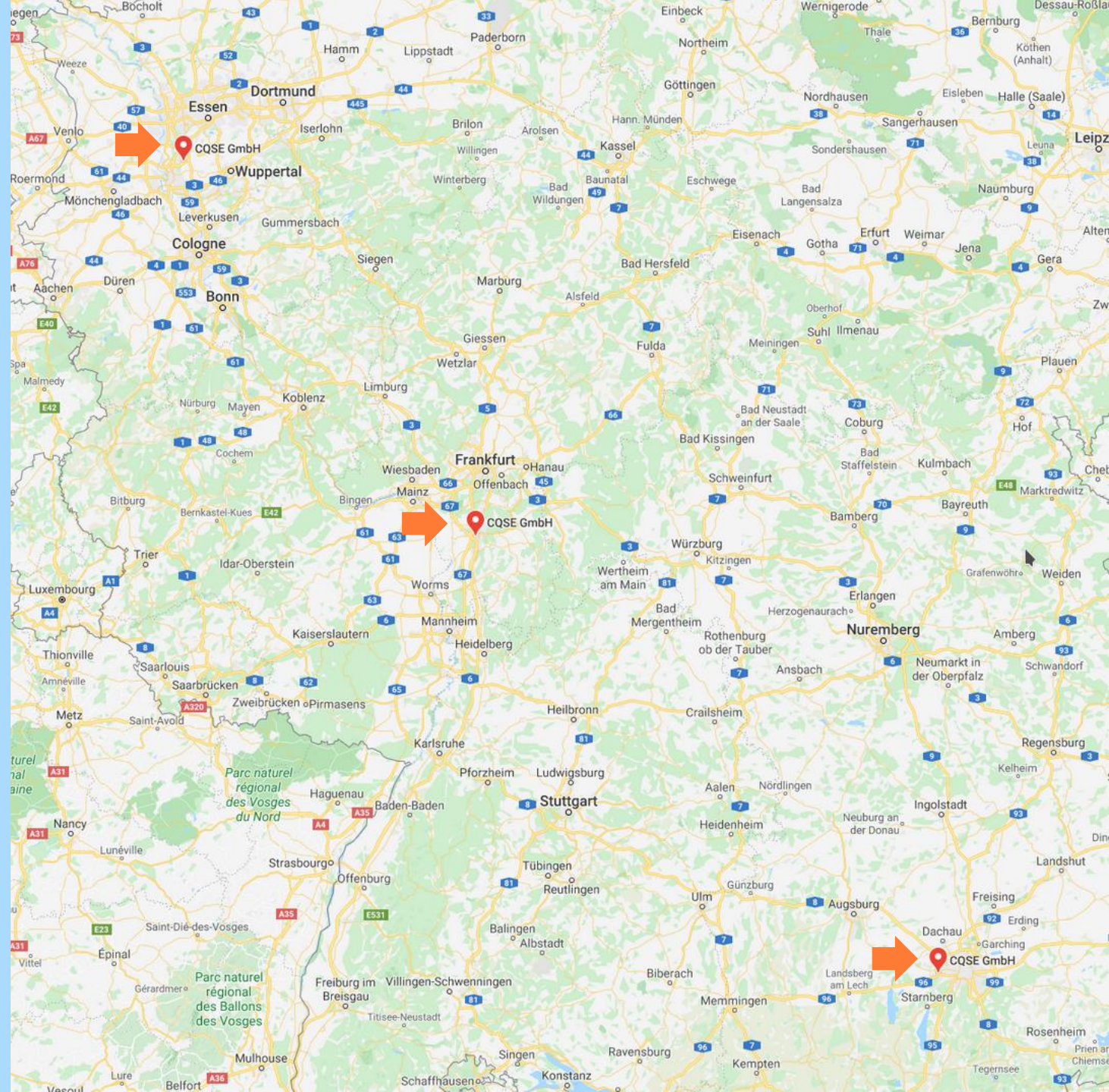
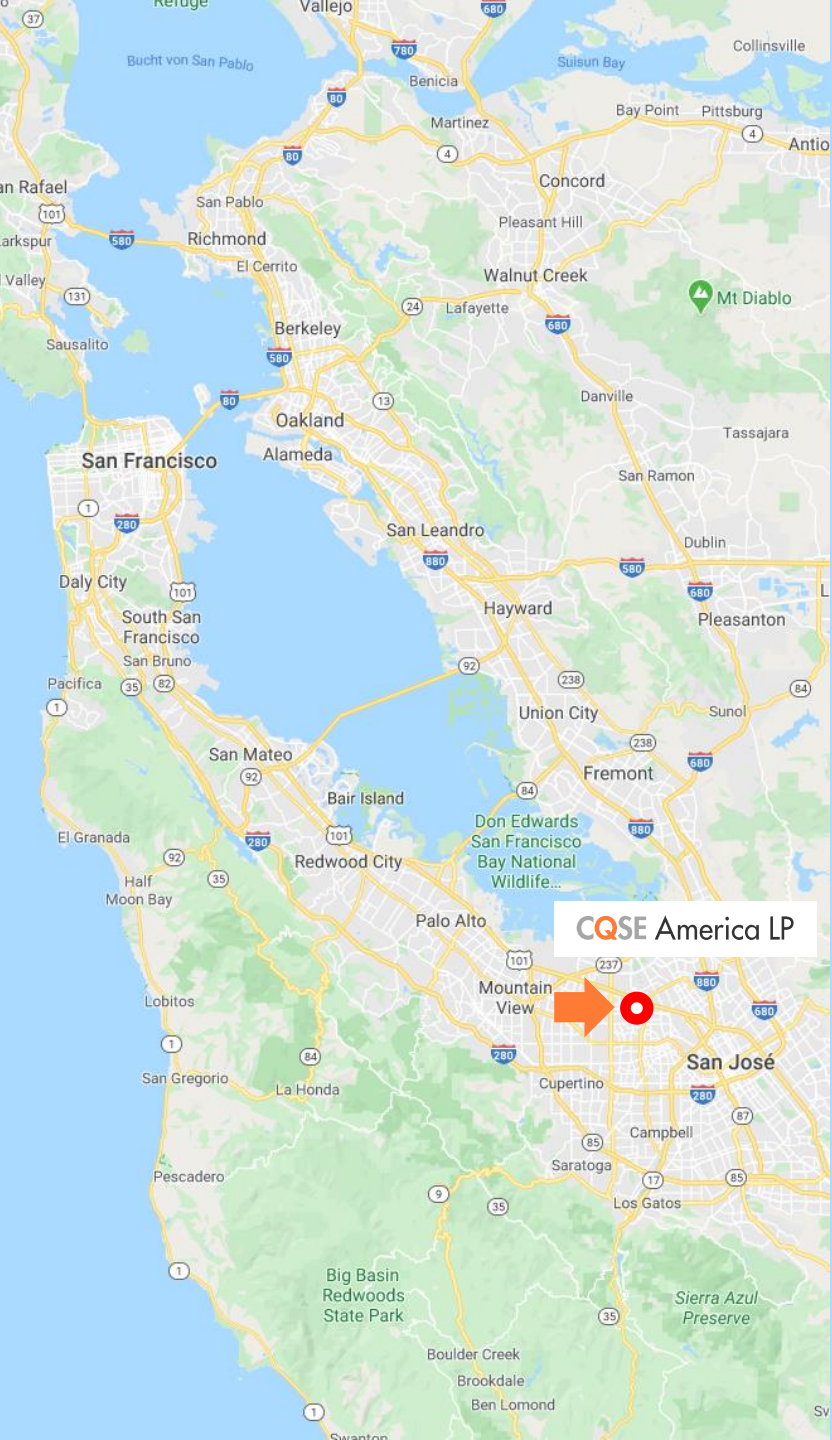
17+ **Promotionen** in  
Software Engineering



Eigene **Forschung**



Enger Kontakt zu  
**Universitäten**





```
28
29 import org.apache.commons.cli.ParseException;
30 import org.slf4j.Logger;
31 import org.slf4j.LoggerFactory;
32
33 /**
34  * JabRef's main class to process command line options and to start the UI
35  */
36 public class JabRefMain extends Application {
37
38     private static final Logger LOGGER = LoggerFactory.getLogger(JabRefMain.class);
39
40     private static String[] arguments;
41
42     public static void main(String[] args) {
43         arguments = args;
44         launch(arguments);
45     }
46
47     @Override
48     public void start(Stage mainStage) {
49         try {
50             // Fail on unsupported Java versions
51             ensureCorrectJavaVersion();
52             FallbackExceptionHandler.installExceptionHandler();
53
54             // Init preferences
55             final JabRefPreferences preferences = JabRefPreferences.getInstance();
56             Globals.prefs = preferences;
57             // Perform migrations
58             PreferencesMigrations.runMigrations();
59
60             configureProxy(preferences.getProxyPreferences());
61
62             Globals.startBackgroundTasks();
63
64             applyPreferences(preferences);
65
```





218.701 LOC

Qualität ?

Technische Schulden ?

Wartbarkeit ?

Zukunftssicherheit ?

Risiken ?

Handlungsbedarf ?



# Statische Analyse

## oder beliebiges anderes Werkzeug



- Dashboard
- Activity
- Findings
- Metrics
- Files**
- Components
- Issues



JabRef

Current Metrics

Path	Files	Lines of Code	Source Lines of Code	Method Length Assessment	Nesting Depth Assessment	Clone Coverage	Comment Completeness Assessment	Findings Density in Code Anomalies	Findings Density in Naming
src	1,821	218,701	140,878	<div style="width: 100%; height: 10px; background-color: green;"></div>	<div style="width: 100%; height: 10px; background-color: green;"></div>	9.6%	<div style="width: 100%; height: 10px; background-color: green;"></div>	5.6	3.1

...the first step in the process of static analysis is to identify the code to be analyzed. This is typically done by specifying a set of source files or a directory structure. Once the code is identified, the next step is to parse it into a representation that can be analyzed. This is often done using a parser that generates an abstract syntax tree (AST) or a similar data structure. The AST represents the hierarchical structure of the code, showing the relationships between different parts of the program. Once the code is represented in this way, various analysis techniques can be applied. For example, control flow graphs (CFGs) can be generated to show the flow of execution through the code. Data flow graphs (DFGs) can be used to track the flow of data through the program. Other techniques include reaching definitions analysis, constant propagation, and dead code elimination. The results of these analyses can be used to identify potential errors, such as null pointer dereferences, buffer overflows, and unreachable code. They can also be used to optimize the code, removing unnecessary operations and improving performance. Finally, the results of the analysis can be presented to the user in a clear and concise manner, often using a graphical interface or a report. This allows the user to understand the findings of the analysis and take appropriate action to address any issues identified.



	Methodenlänge	Schachtelung	Klonüberdeckung	Kommentierung	Codeanomalien pro 1.000 SLOC	Naming-Verstöße pro 1.000 SLOC
JabRef			9,6 %	22,8 %	5,6	3,1

Methodenlänge

Schachtelung

Klonüberdeckung

Kommentierung

Codeanomalien  
pro 1.000 SLOC

Naming-Verstöße  
pro 1.000 SLOC

JabRef



9,6%

22,8%

5,6

3,1

```

420
427 private List<String> refreshCiteMarkersInternal(List<BibDatabase> databases, OOBibStyle style)
428 throws WrappedTargetException, IllegalArgumentException, NoSuchElementException,
429         UndefinedCharacterFormatException, UnknownPropertyException, PropertyVetoException,
430         CreationException, BibEntryNotFoundException {
431     for (int i = 0; i < names.size(); i++) {
432         Matcher citeMatcher = CITE_PATTERN.matcher(names.get(i));
433         List<String> cited = findCitedKeys();
434         Map<String, BibDatabase> linkSourceBase = new HashMap<>();
435         Map<BibEntry, BibDatabase> entries = findCitedEntries();
436         XNameAccess xReferenceMarks = getReferenceMarks();
437         List<String> names;
438         // We need to sort the reference marks according to
439         names = sortedReferenceMarks;
440         // We need to sort the reference marks according to
441         // entries
442         SortedMap<BibEntry, BibDatabase> newMap = new
443         SortedMap<BibEntry, BibDatabase> (BibDatabase::compareTo);
444         // We need to sort the reference marks according to
445         SortedMap<BibEntry, BibDatabase> newMap = new
446         SortedMap<BibEntry, BibDatabase> (BibDatabase::compareTo);
447         newMap.put(bibtexEntryBibDatabaseEntry);
448         entries = newMap;
449         // Rebuild the list of cited keys according to
450         cited.clear();
451         for (BibEntry entry : entries.keySet()) {
452             cited.add(entry.getCiteKeyOptional().orElse(
453             entry.getCiteKey()));
454         }
455         names = Arrays.asList(xReferenceMarks.getElem
456         names = sortedReferenceMarks;
457         // Remove all reference marks that don't look like
458         List<String> tmp = new ArrayList<>();
459         for (String name : names) {
460             if (CITE_PATTERN.matcher(name).find()) {
461                 tmp.add(name);
462             }
463         }
464         names = tmp;
465         Map<String, Integer> numbers = new HashMap<>();
466         int lastNum = 0;
467         // First compute citation markers for all citation
468         String[] citMarkers = new String[names.size()];
469         String[] normCitMarkers = new String[names.size()];
470         String[][] bibTexKeys = new String[names.size()][];
471         int nInGroupingCount = style.getInt(CIT_PROPERTY_000
472         int[] types = new int[names.size()];
473
474     for (int i = 0; i < names.size(); i++) {
475         for (int j = 0; j < keys.length; j++) {
476             String typeStr = citeMatcher.group(1);
477             int type = Integer.parseInt(typeStr);
478             String[] keys = citeMatcher.group(2).split(
479             bibTexKeys[i] = keys;
480             BibEntry[] cEntries = new BibEntry[keys.l
481             for (int j = 0; j < cEntries.length; j++)
482             BibDatabase database = linkSourceBase.
483             Optional<BibEntry> tmpEntry = optional
484             if (database != null) {
485                 tmpEntry = database.getEntryByKey
486             }
487             if (tmpEntry.isPresent()) {
488                 cEntries[j] = tmpEntry.get();
489             } else {
490                 LOGGER.info("Citation key not found
491                 throw new BibEntryNotFoundException(
492                 LOGGER.info("Problem with referenc
493                 throw new BibEntryNotFoundException(
494                 LOGGER.info("Could not resolve c
495             }
496             String[] normCitMarker = new String[keys.l
497             String citationMarker;
498             if (style.isCitationKeyCiteMarkers()) {
499                 StringBuilder sb = new StringBuilder(
500                 normCitMarkers[i] = new String[keys.l
501                 for (int j = 0; j < keys.length; j++)
502                 normCitMarkers[i][j] = cEntries[j].
503                 sb.append(cEntries[j].getCiteKeyOp
504                 if (j < (keys.length - 1)) {
505                     sb.append(", ");
506                 }
507                 citationMarker = sb.toString();
508             } else if (style.isNumberEntries()) {
509                 // We have sorted the citation mar
510                 // so we simply count up for each
511                 List<Integer> num = new ArrayList<>();
512                 for (int j = 0; j < keys.length; j
513                 if (cEntries[j] instanceof U
514                     num.add(j, -1);
515                 num.add(j, lastNum + 1);
516                 if (numbers.containsKey(k
517                     num.set(j, numbers.get
518             }
519             citMarkers[i] = citationMarker;
520             normCitMarkers[i] = normCitMarker;
521             uniqueifiers.clear();
522             // See if there are duplicate citations marks
523             // use uniqueifiers:
524             Map<String, List<String>> refKeys = new HashM
525             Map<String, List<Integer>> refNums = new HashM
526             for (int i = 0; i < citMarkers.length; i++) {
527                 String[] markers = markers[i];
528                 for (int j = 0; j < markers.length; j++)
529                 if (refKeys.containsKey(marker)) {
530                     refKeys.get(marker).add(curren
531                     refNums.get(marker).add(curre
532                 } else {
533                     List<String> l = new ArrayList<>();
534                     l.add(currentKey);
535                     List<Integer> l2 = new ArrayList<
536                     l2.add(i);
537                     refNums.put(marker, l2);
538                 }
539             }
540             // Go through the collected lists and see whe
541             for (Map.Entry<String, List<String>> stringLi
542                 List<String> keys = stringListEntry.getVal
543                 if (keys.size() > 1) {
544                     // This marker appears for more than
545                     for (String key : keys) {
546                         // Update the map of uniqueifiers
547                         // citation markers, and for the
548                         uniqueifiers.put(key, string.valu
549                         uniqueifiers.put(key, string.valu
550                     }
551                 }
552             }
553             // Finally, go through all citation markers,
554
555     int maxAuthorsFirst = style.getInt(CIT_PROPERTY_000BIBSTYLE_MAX_AUTHORS_FIRST);
556     Set<String> seenBefore = new HashSet<>();
557     for (int j = 0; j < bibTexKeys.length; j++) {
558         boolean needsChange = false;
559         int[] firstLinAuthors = new int[bibTexKeys
560         String[] uniqif = new String[bibTexKeys[i]
561         BibEntry[] cEntries = new BibEntry[bibTex
562         for (int k = 0; k < bibTexKeys[i].length;
563             String currentKey = bibTexKeys[i][k];
564             firstLinAuthors[k] = 1;
565             if (maxAuthorsFirst > 0) {
566                 if (!seenBefore.contains(currentKey
567                     firstLinAuthors[k] = maxAuthor
568                 seenBefore.add(currentKey);
569             }
570             String uniq = uniqueifiers.get(currentK
571             Optional<BibEntry> tmpEntry = optional
572             if (uniq == null) {
573                 if (firstLinAuthors[k] > 0) {
574                     BibDatabase database = linkSou
575                     if (database != null) {
576                         tmpEntry = database.getEnt
577                     } else {
578                         BibDatabase database = linkSou
579                         if (database != null) {
580                             tmpEntry = database.getEnt
581                         }
582                     }
583                     uniqif[k] = "";
584                 } else {
585                     needsChange = true;
586                     BibDatabase database = linkSource
587                     if (database != null) {
588                         tmpEntry = database.getEntryBy
589                     }
590                     uniqif[k] = uniq;
591                 }
592             }
593             if (needsChange) {
594                 BibDatabase database = linkSource
595                 if (database != null) {
596                     tmpEntry = database.getEntryBy
597                 }
598                 uniqif[k] = uniq;
599             }
600             if (tmpEntry.isPresent()) {
601                 cEntries[k] = tmpEntry.get();
602             }
603             if (needsChange) {
604                 if (style.getCitationMarke
605                     types[i] = OOBibDatabase.AUTHOR
606             }
607         }
608     }
609     // Refresh all reference marks with the citation markers we computed:
610     boolean hadBibSection = getBookmarkRange(OOBibDatabase.BIB_SECTION_NAME) != null;
611     // Check if we are supposed to set a character format for citations:
612     boolean mustTestCharFormat = style.isFormatCitations();
613     for (int i = 0; i < names.size(); i++) {
614         Object referenceMark = xReferenceMarks.getByEntryName(names.get(i));
615         XTextContent bookmark = UnoRuntime.queryInterface(XTextContent.class, referenceMark);
616         XTextCursor cursor = bookmark.getAnchor().getText().createTextCursorByRange(bookmark.getAnchor());
617         if (mustTestCharFormat) {
618             // If we are supposed to set character format for citations, must run a test before we
619             // delete old citation markers. Otherwise, if the specified character format doesn't
620             // exist, we end up deleting the markers before the process crashes due to a missing
621             // format, with catastrophic consequences for the user.
622             mustTestCharFormat = false; // need to do this only once
623             XPropertySet xCursorProps = UnoRuntime.queryInterface(XPropertySet.class, cursor);
624             String charStyle = style.getCitationCharacterFormat();
625             try {
626                 xCursorProps.setPropertyValue(CHAR_STYLE_NAME, charStyle);
627             } catch (UnknownPropertyException | PropertyVetoException | IllegalArgumentException |
628             WrappedTargetException ex) {
629                 throw new UndefinedCharacterFormatException(charStyle);
630             }
631         }
632         text.removeTextContent(bookmark);
633         insertReferenceMark(names.get(i), citMarkers[i], cursor, types[i] != OOBibDatabase.INVISIBLE_CIT, style
634         if (hadBibSection && (getBookmarkRange(OOBibDatabase.BIB_SECTION_NAME) == null)) {
635             // We have overwritten the marker for the start of the reference list.
636             // We need to add it again.
637             cursor.collapsateToEnd();
638             OOUTil.insertParagraphBreak(text, cursor);
639             insertBookmark(OOBibDatabase.BIB_SECTION_NAME, cursor);
640         }
641     }
642     List<String> unresolvedKeys = new ArrayList<>();
643     for (BibEntry entry : entries.keySet()) {
644         if (entry instanceof UndefinedBibEntry) {
645             String key = (UndefinedBibEntry) entry).getKey();
646             if (unresolvedKeys.contains(key)) {
647                 unresolvedKeys.add(key);
648             }
649         }
650     }
651     return unresolvedKeys;
652 }

```

Methodenlänge

Schachtelung

Klonüberdeckung

Kommentierung

Codeanomalien  
pro 1.000 SLOC

Naming-Verstöße  
pro 1.000 SLOC

JabRef



9,6%

22,8%

5,6

3,1

```
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
```

1 2 3

```
4 for (int k = 0; k < tokens.length; k++) {
    String token = tokens[k];
    if (abbreviateThatIsSingleLetter) {
        String[] dashes = token.split("-");

        token = Arrays.asList(dashes).stream().map(BibtexNameFormatter::getFirstCharOfString)
            .collect(Collectors.joining("-"));
    }

    // Output token
    sb.append(token);

5 if (k < (tokens.length - 1)) {
    // Output Intertoken String
6 if (interToken == null) {
7 if (abbreviateThatIsSingleLetter) {
8 sb.append('.');
    }
    // No clue what this means (What the hell are tokens anyway???)
    // if (lex_class[name_sep_char[cur_token]] = sep_char) then
    // append ex_buf_char and check (name_sep_char[cur_token])
```



Methodenlänge

Schachtelung

Klonüberdeckung

Kommentierung

Codeanomalien  
pro 1.000 SLOC

Naming-Verstöße  
pro 1.000 SLOC

JabRef



9,6 %

22,8 %

5,6

3,1

```
125 }
126
127 private static void addResourceFile(String name, String resource, ZipOutput
128     ZipEntry zipEntry = new ZipEntry(name);
129     out.putNextEntry(zipEntry);
130     OpenDocumentSpreadsheetCreator.addFromResource(resource, out);
131     out.closeEntry();
132 }
133
134 private static void addFromResource(String resource, OutputStream out) {
135     URL url = OpenDocumentSpreadsheetCreator.class.getResource(resource);
136     try (InputStream in = url.openStream()) {
137         byte[] buffer = new byte[256];
138         synchronized (out) {
139             while (true) {
140                 int bytesRead = in.read(buffer);
141                 if (bytesRead == -1) {
142                     break;
143                 }
144                 out.write(buffer, 0, bytesRead);
145             }
146         }
147     } catch (IOException e) {
148         LOGGER.warn("Cannot get resource", e);
149     }
150 }
151 }
152
```

```
105 }
106
107 private static void addResourceFile(String name, String resource, ZipOutput
108     ZipEntry zipEntry = new ZipEntry(name);
109     out.putNextEntry(zipEntry);
110     OpenOfficeDocumentCreator.addFromResource(resource, out);
111     out.closeEntry();
112 }
113
114 private static void addFromResource(String resource, OutputStream out) {
115     URL url = OpenOfficeDocumentCreator.class.getResource(resource);
116     try (InputStream in = url.openStream()) {
117         byte[] buffer = new byte[256];
118         synchronized (out) {
119             while (true) {
120                 int bytesRead = in.read(buffer);
121                 if (bytesRead == -1) {
122                     break;
123                 }
124                 out.write(buffer, 0, bytesRead);
125             }
126         }
127     } catch (IOException e) {
128         LOGGER.warn("Cannot get resource", e);
129     }
130 }
131 }
132
```



	Methodenlänge	Schachtelung	Klonüberdeckung	Kommentierung	Codeanomalien pro 1.000 SLOC	Naming-Verstöße pro 1.000 SLOC
JabRef			9,6%	22,8%	5,6	3,1

```

9  import org.jabref.logic.tion.Localization;
10 import org.jabref.model.search.rules.SentenceAnalyzer;
11
12 public class ContainsAndRegexBasedSearchRuleDescriber implements SearchDescriber {
13
14     private final boolean regExp;
15     private final boolean caseSensitive;
16     private final String query;
17
18     public ContainsAndRegexBasedSearchRuleDescriber(boolean caseSensitive, boolean regExp, String query) {
19         this.caseSensitive = caseSensitive;
20         this.regExp = regExp;
21         this.query = query;
22     }
23
24     @Override
25     public TextFlow getDescription() {
26         List<String> words = new SentenceAnalyzer(query).getWords();
27         String firstWord = words.isEmpty() ? "" : words.get(0);
28
29         String temp = regExp ? Localization.lang(

```

Methodenlänge

Schachtelung

Klonüberdeckung

Kommentierung

Codeanomalien  
pro 1.000 SLOC

Naming-Verstöße  
pro 1.000 SLOC

JabRef



9,6%

22,8%

5,6

3,1

```
437
438 List<String> names;
439 if (style.isSortByPosition()) {
440     // We need to sort the reference marks according to their
441     names = sortedReferenceMarks;
442 } else if (style.isNumberEntries()) {
443     // We need to sort the reference marks according to the s
444     // entries:
445     SortedMap<BibEntry, BibDatabase> newMap = new TreeMap<>(e
446     for (Map.Entry<BibEntry, BibDatabase> bibtexEntryBibtexDat
447         newMap.put(bibtexEntryBibtexDatabaseEntry.getKey(), b
448     }
449     entries = newMap;
450     // Rebuild the list of cited keys according to the sort o
451     cited.clear();
452     for (BibEntry entry : entries.keySet()) {
453         cited.add(entry.getCiteKeyOptional().orElse(null));
454     }
455     names = Arrays.asList(xReferenceMarks.getElementNames());
456 } else {
457     names = sortedReferenceMarks;
458 }
459
460 // Remove all reference marks that don't look like JabRef cit
```



```
323
324 * Sets the title of the main window.
325 */
326 public void setWindowTitle() {
327     BasePanel panel = getCurrentBasePanel();
328
329     // no database open
330     if (panel == null) {
331         // setTitle(FRAME_TITLE);
332         return;
333     }
334
335     String mode = panel.getBibDatabaseContext().getMode().getFormattedName();
336     String modeInfo = String.format(" (%s)", Localization.lang("%0 mode", mode));
337     boolean isAutosaveEnabled = Globals.prefs.getBoolean(JabRefPreferences.LOCAL_
338
339     if (panel.getBibDatabaseContext().getLocation() == DatabaseLocation.LOCAL) {
340         String changeFlag = panel.isModified() && !isAutosaveEnabled ? "*" : "";
341         String databaseFile = panel.getBibDatabaseContext()
342             .getDatabasePath()
343             .map(Path::toString)
344             .orElse(Localization.lang("untitled"));
345         // setTitle(FRAME_TITLE + " - " + databaseFile + changeFlag + modeInfo);
346     } else if (panel.getBibDatabaseContext().getLocation() == DatabaseLocation.SHA
347         // setTitle(FRAME_TITLE + " - " + panel.getBibDatabaseContext().getDBMSFu
```



	Methodenlänge	Schachtelung	Klonüberdeckung	Kommentierung	Codeanomalien pro 1.000 SLOC	Naming-Verstöße pro 1.000 SLOC
JabRef			9,6 %	22,8 %	5,6	3,1

```

9 public class UndoableModifySubtree extends AbstractUndoableJabRefEdit {
10
11     /**
12      * A backup of the groups before the modification
13      */
14     private final GroupTreeNode m_groupRoot;
15
16     private final GroupTreeNode m_subtreeBackup;
17
18     /**
19      * The path to the global groups root node
20      */
21     private final List<Integer> m_subtreeRootPath;
22
23     /**
24      * This holds the new subtree (the root's modified children) to allow redo.
25      */
26     private final List<GroupTreeNode> m_modifiedSubtree = new ArrayList<>();

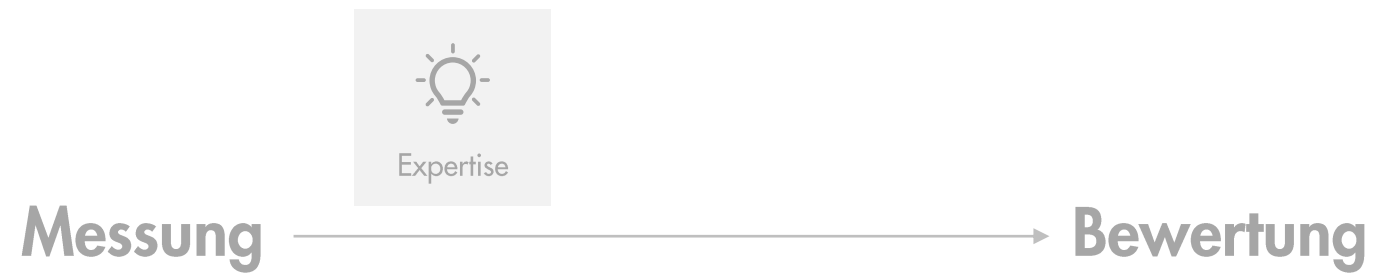
```



	Methodenlänge	Schachtelung	Klonüberdeckung	Kommentierung	Codeanomalien pro 1.000 SLOC	Naming-Verstöße pro 1.000 SLOC
JabRef			9,6 %	22,8 %	5,6	3,1

Handlungsbedarf ?

**Messung ≠ Bewertung !**



The first rule of functions is that they should be small. The second rule of functions is that *they should be smaller than that*. [...] **Functions should hardly ever be 20 lines long.**

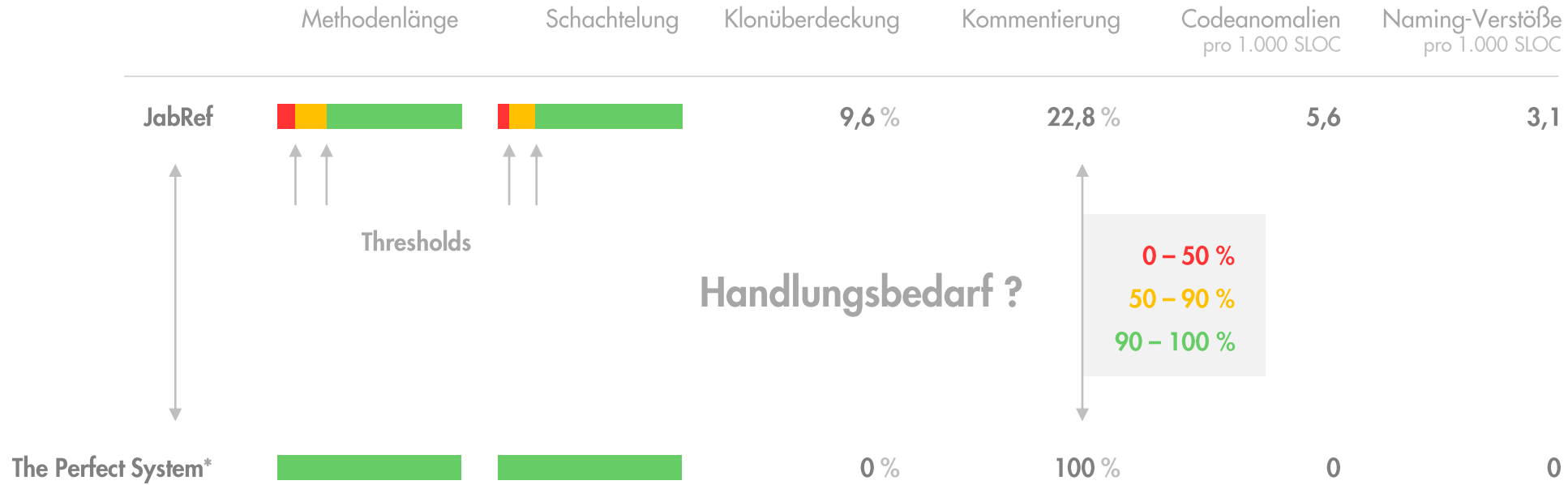
– Robert C. Martin, 2008

**Number one in the stink parade is duplicated code.** If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them.

– Martin Fowler, 1999

**Naming conventions make programs more understandable** by making them easier to read. They can also give information about the function of the identifier [...]

– Java Code Conventions, 1997



\* Existiert nicht

» Warum sind Methoden ab 30 Zeilen zu lang und nicht erst ab 50? «



Expertise



Thresholds

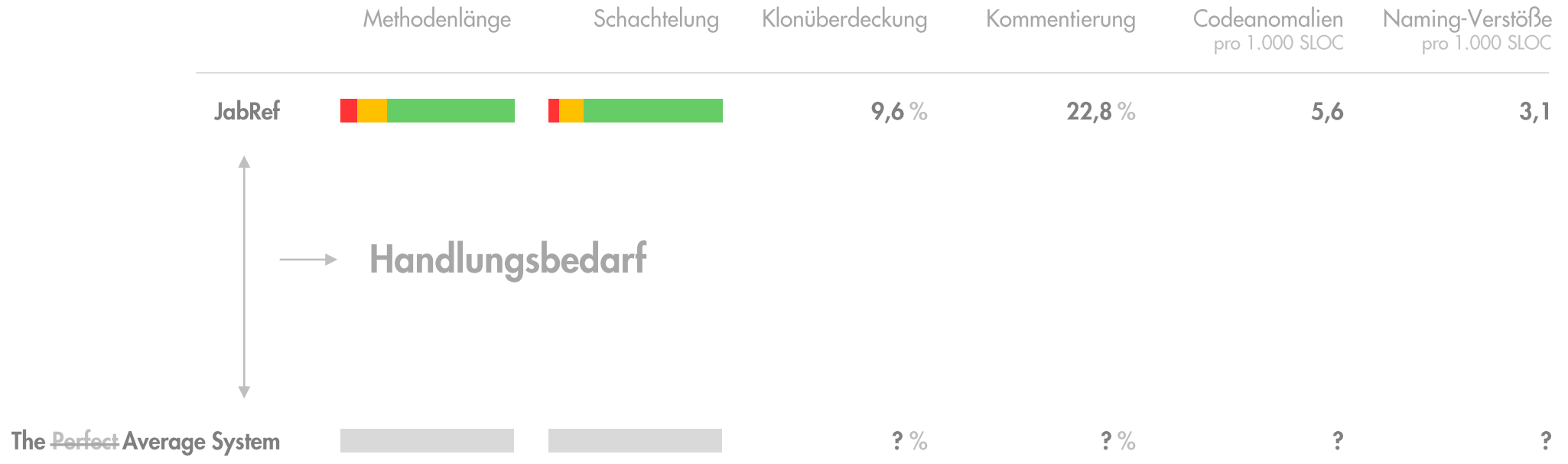


Benchmark

Messung



Bewertung

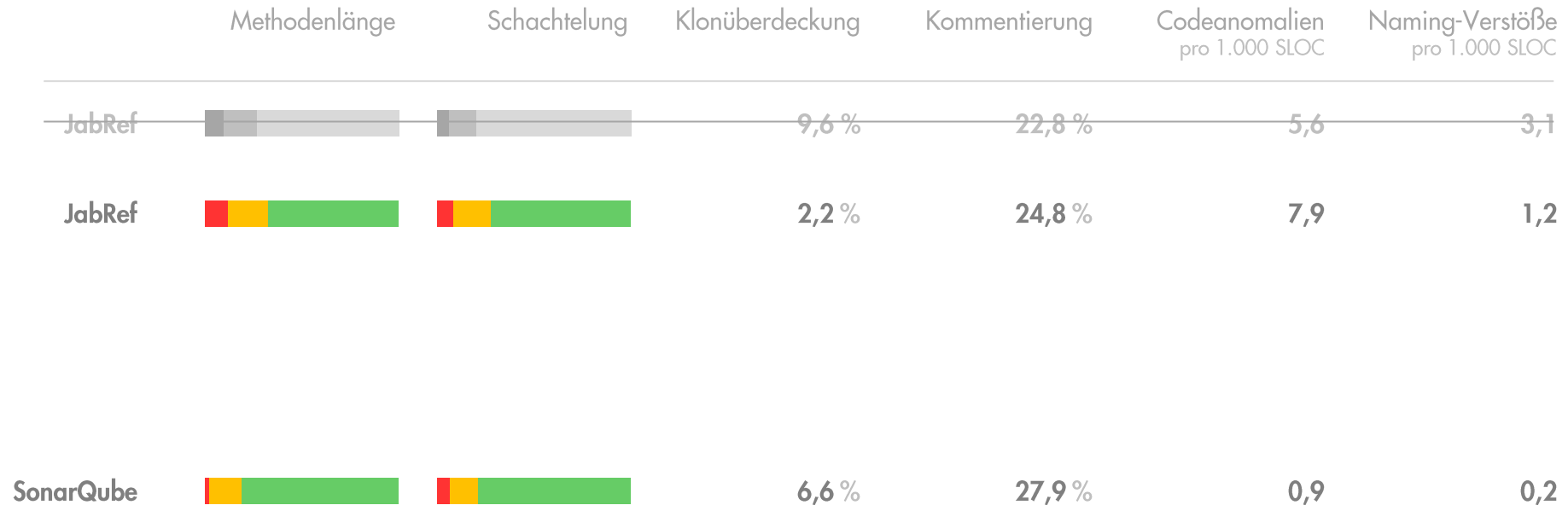


» Ja, aber ... «

	Methodenlänge	Schachtelung	Klonüberdeckung	Kommentierung	Codeanomalien pro 1.000 SLOC	Naming-Verstöße pro 1.000 SLOC
JabRef		 ★	9,6 %	22,8 %	5,6	3,1
SonarQube	★ 		★ 6,6 %	★ 27,9 %	★ 0,9	★ 0,2

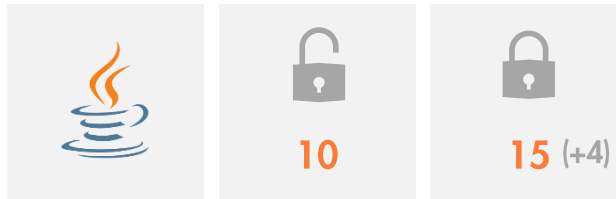
# Klonüberdeckung pro Datei in JabRef



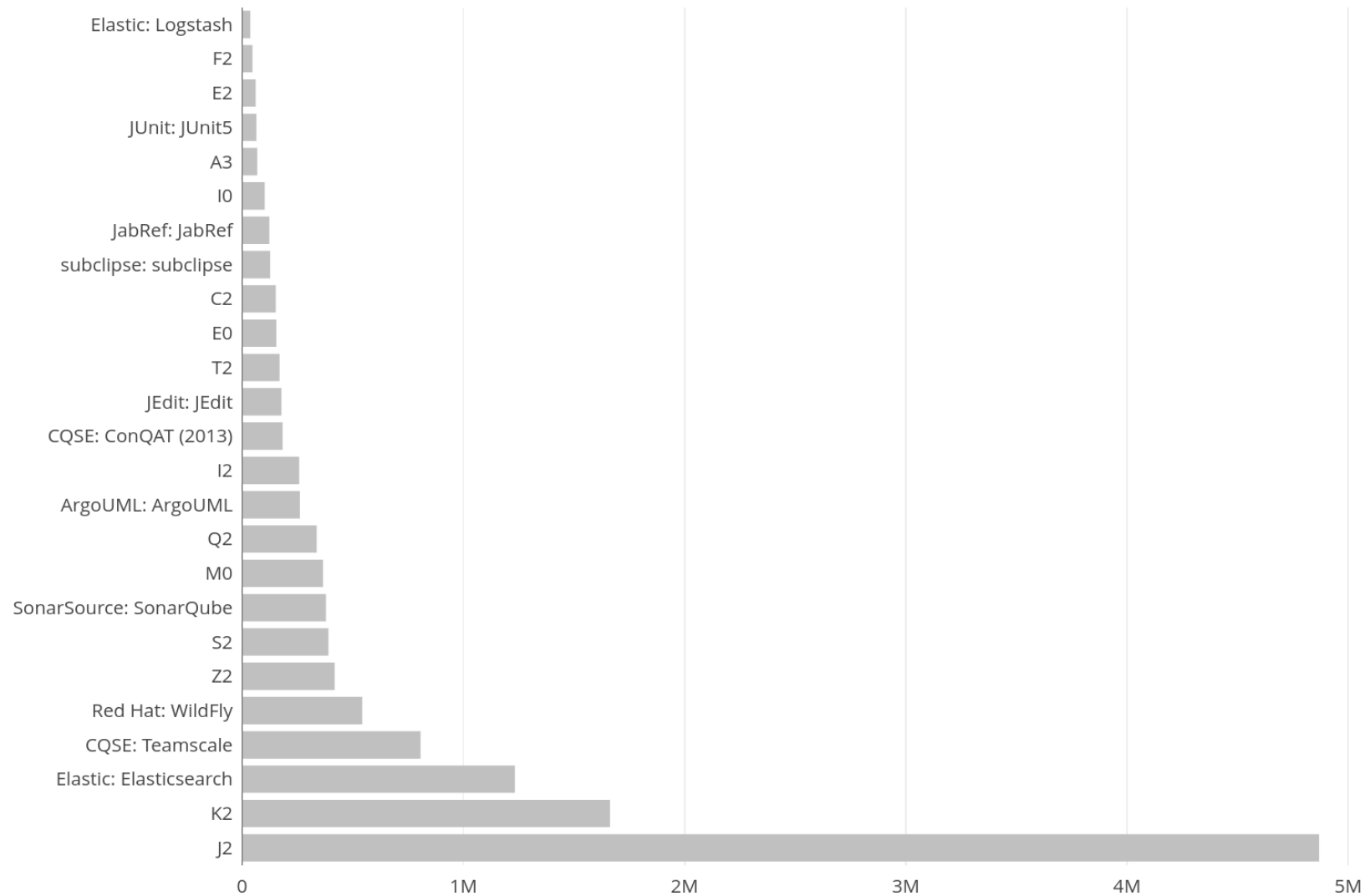




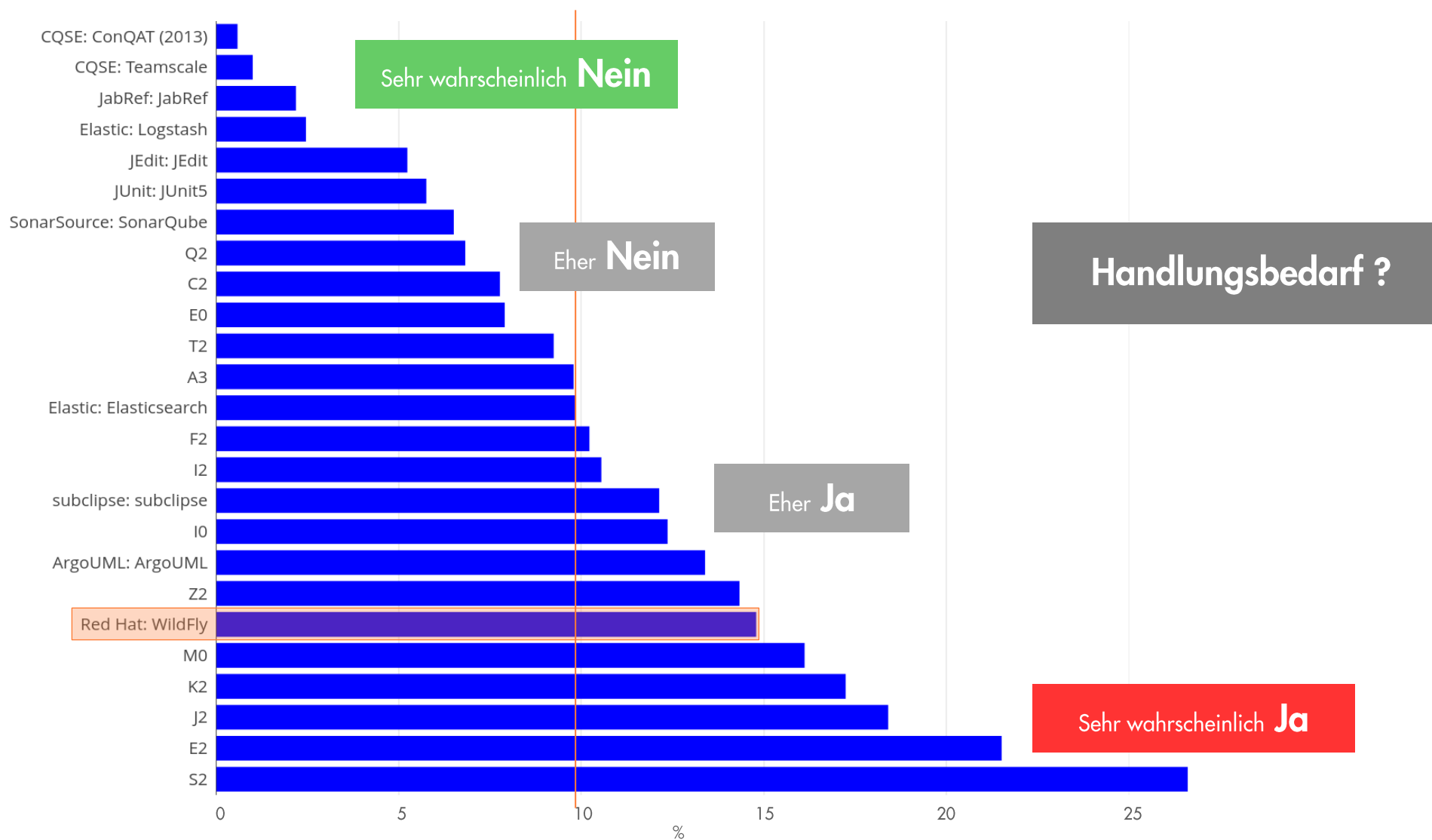
## CQSE-Benchmark



# Lines of Code



# Klonüberdeckung



A treemap visualization showing the structure of a code repository. The rectangles are colored in shades of blue, with some highlighted in orange. Labels are overlaid on the treemap, identifying specific directories and files. The labels are: src/clustering/ee/cache/.../scheduler/FastConcurrentDirectDeque.java, src/clustering/ee/cache/.../scheduler/PortableConcurrentDirectDeque.java, src/security/subsystem/.../lru/FastConcurrentDirectDeque.java, src/security/subsystem/.../lru/PortableConcurrentDirectDeque.java, src/jpa/hibernate4\_1, src/jpa/hibernate5\_3, src/jpa/hibernate4\_3, and src/jpa/hibernate5. The treemap shows a hierarchical structure with many small rectangles and some larger ones, indicating a complex codebase.

src/clustering/ee/cache/.../scheduler/FastConcurrentDirectDeque.java

src/clustering/ee/cache/.../scheduler/PortableConcurrentDirectDeque.java

src/security/subsystem/.../lru/FastConcurrentDirectDeque.java

src/security/subsystem/.../lru/PortableConcurrentDirectDeque.java

src/jpa/hibernate4\_1

src/jpa/hibernate5\_3

src/jpa/hibernate4\_3

src/jpa/hibernate5

## src/jpa/hibernate4\_1

```

73
74
75     private org.hibernate.stat.Statistics getBaseStatistics(EntityManagerFactory entityMa
76         if (entityManagerFactory == null) {
77             return null;
78         }
79         HibernateEntityManagerFactory entityManagerFactoryImpl = (HibernateEntityManagerF
80         SessionFactory sessionFactory = entityManagerFactoryImpl.getSessionFactory();
81         if (sessionFactory != null) {
82             return sessionFactory.getStatistics();
83         }
84         return null;
85     }
86
87     private org.hibernate.stat.EntityStatistics getStatistics(EntityManagerFactory entity
88     HibernateEntityManagerFactory entityManagerFactoryImpl = (HibernateEntityManagerF
89     SessionFactory sessionFactory = entityManagerFactoryImpl.getSessionFactory();
90     if (sessionFactory != null) {
91         return sessionFactory.getStatistics().getEntityStatistics(entityName);
92     }
93     return null;
94 }
95
96 private Operation entityDeleteCount = new Operation() {

```

## src/jpa/hibernate4\_3

```

73
74
75     private org.hibernate.stat.Statistics getBaseStatistics(EntityManagerFactory entityMa
76         if (entityManagerFactory == null) {
77             return null;
78         }
79         HibernateEntityManagerFactory entityManagerFactoryImpl = (HibernateEntityManagerF
80         SessionFactory sessionFactory = entityManagerFactoryImpl.getSessionFactory();
81         if (sessionFactory != null) {
82             return sessionFactory.getStatistics();
83         }
84         return null;
85     }
86
87     private org.hibernate.stat.EntityStatistics getStatistics(EntityManagerFactory entity
88     if (entityManagerFactory == null) {
89         return null;
90     }
91     HibernateEntityManagerFactory entityManagerFactoryImpl = (HibernateEntityManagerF
92     SessionFactory sessionFactory = entityManagerFactoryImpl.getSessionFactory();
93     if (sessionFactory != null) {
94         return sessionFactory.getStatistics().getEntityStatistics(entityName);
95     }
96     return null;
97 }
98
99 private Operation entityDeleteCount = new Operation() {

```

## src/jpa/hibernate5

```

67
68
69     private org.hibernate.stat.Statistics getBaseStatistics(EntityManagerFactory entityMa
70         if (entityManagerFactory == null) {
71             return null;
72         }
73         SessionFactory sessionFactory = entityManagerFactory.unwrap(SessionFactory.class)
74         if (sessionFactory != null) {
75             return sessionFactory.getStatistics();
76         }
77         return null;
78     }
79
80     private org.hibernate.stat.EntityStatistics getStatistics(EntityManagerFactory entity
81     if (entityManagerFactory == null) {
82         return null;
83     }
84     SessionFactory sessionFactory = entityManagerFactory.unwrap(SessionFactory.class)
85     if (sessionFactory != null) {
86         return sessionFactory.getStatistics().getEntityStatistics(pathAddress.getValu
87     }
88     return null;
89 }
90
91 private Operation entityDeleteCount = new Operation() {

```

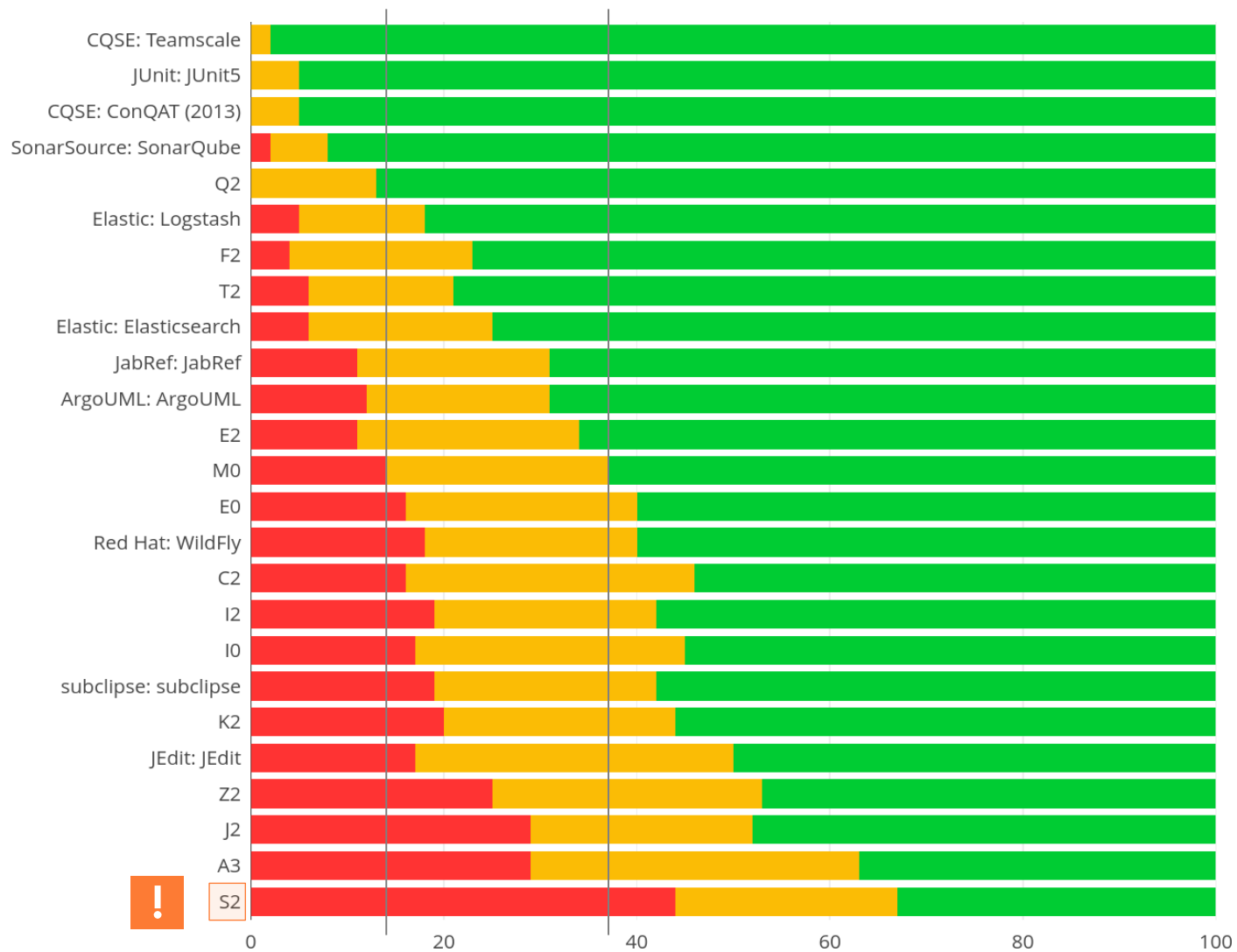
## src/jpa/hibernate5\_3

```

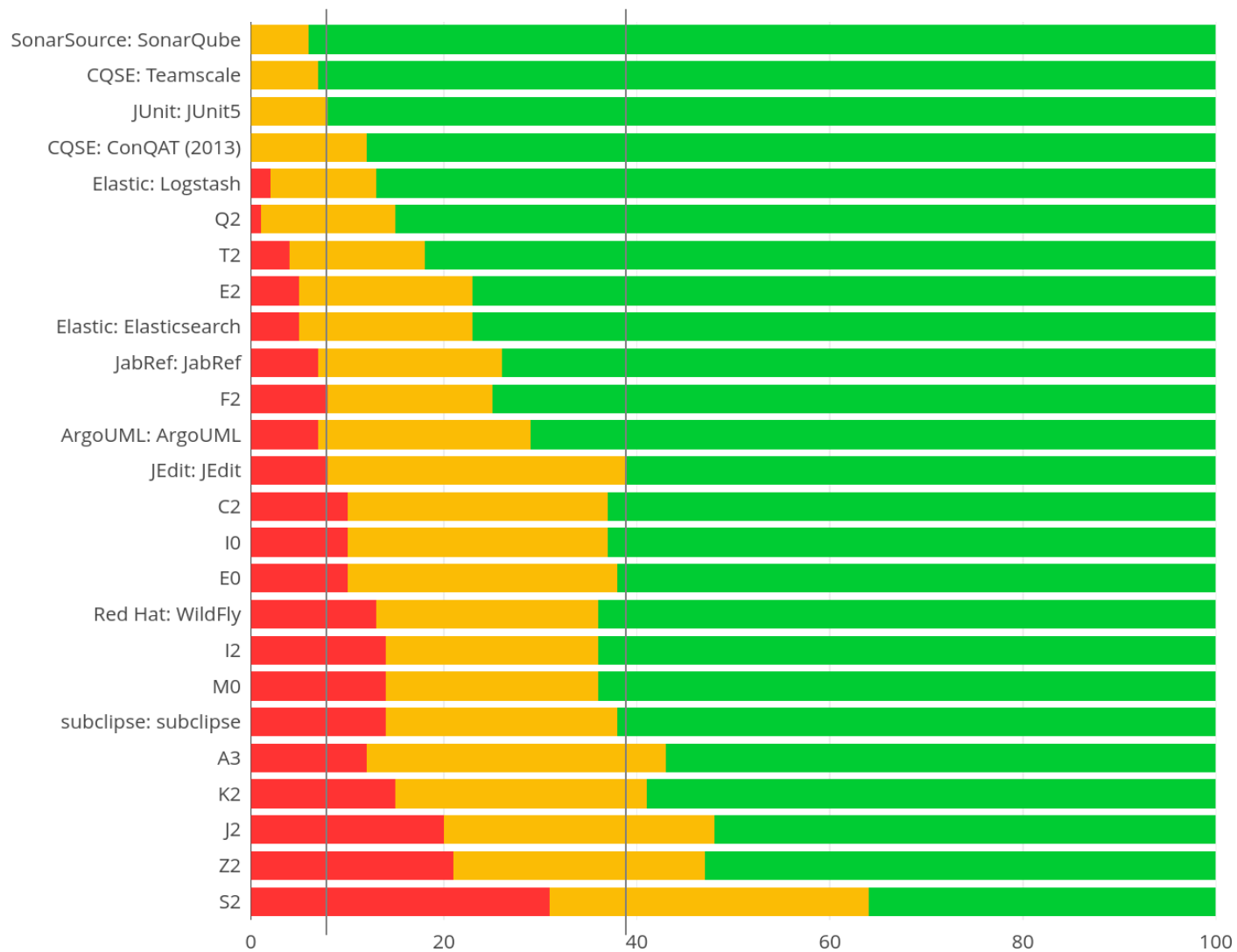
67
68
69     private org.hibernate.stat.Statistics getBaseStatistics(EntityManagerFactory entityMa
70         if (entityManagerFactory == null) {
71             return null;
72         }
73         SessionFactory sessionFactory = entityManagerFactory.unwrap(SessionFactory.class)
74         if (sessionFactory != null) {
75             return sessionFactory.getStatistics();
76         }
77         return null;
78     }
79
80     private org.hibernate.stat.EntityStatistics getStatistics(EntityManagerFactory entity
81     if (entityManagerFactory == null) {
82         return null;
83     }
84     SessionFactory sessionFactory = entityManagerFactory.unwrap(SessionFactory.class)
85     if (sessionFactory != null) {
86         return sessionFactory.getStatistics().getEntityStatistics(pathAddress.getValu
87     }
88     return null;
89 }
90
91 private Operation entityDeleteCount = new Operation() {

```

# Methodenlänge

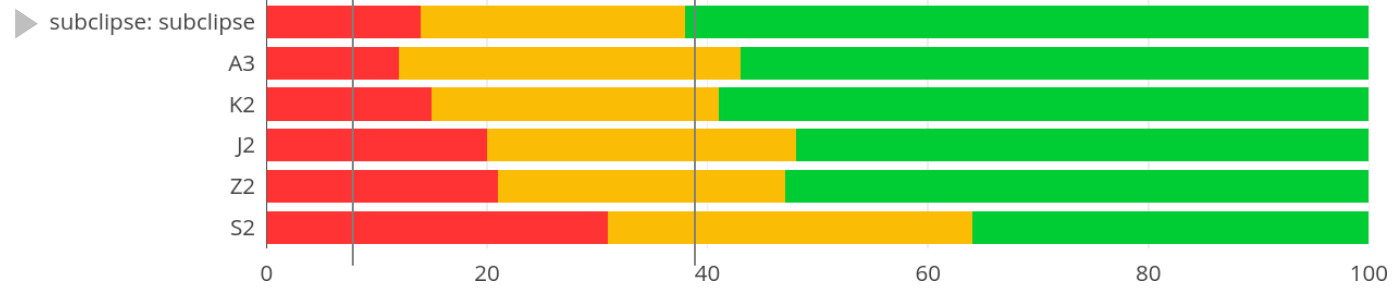


# Schachtelungstiefe



So

```
313 showCompareButton.setSelection(showCompare);
314 1 2 3 showCompareButton.addSelectionListener(
315     4 new SelectionListener(){
316     5 public void widgetSelected(SelectionEvent e) {
317     6 showComparePane(!showCompare);
318     7 if (showCompare) {
319     7 IStructuredSelection selection = (IStructuredSelection)resourceSelectionTree.getTreeView().getSelection();
320     8 if (!selection.isEmpty()) {
321     8 Object sel0 = selection.getFirstElement();
322     9 if (sel0 instanceof IFile) {
323     9 final ISVNLocalResource localResource= SVNWorkspaceRoot.getSVNResourceFor((IFile)sel0);
324     9 try {
325     10 // if any alternate compare actions are defined from the extension point
326     10 // then call those actions instead of showing the default compare dialog
327     11 if (alternateCompareActions.length > 0) {
328     11 StructuredSelection localResourceSelection = new StructuredSelection(localResource);
329     12 for (int i = 0; i < alternateCompareActions.length; i++) {
330     12 // make sure the selection is up to date
331     12 alternateCompareActions[i].selectionChanged(localResourceSelection);
332     12 alternateCompareActions[i].run();
333     12 }
334     12 } else {
335     12 setCompareInput(new SVNLocalCompareInput(localResource, SVNRevision.BASE, true));
336     12 }
337     12 } catch (Exception e1) {
338     12 }
339     12 }
340     12 }
341     12 }
342     12 }
```

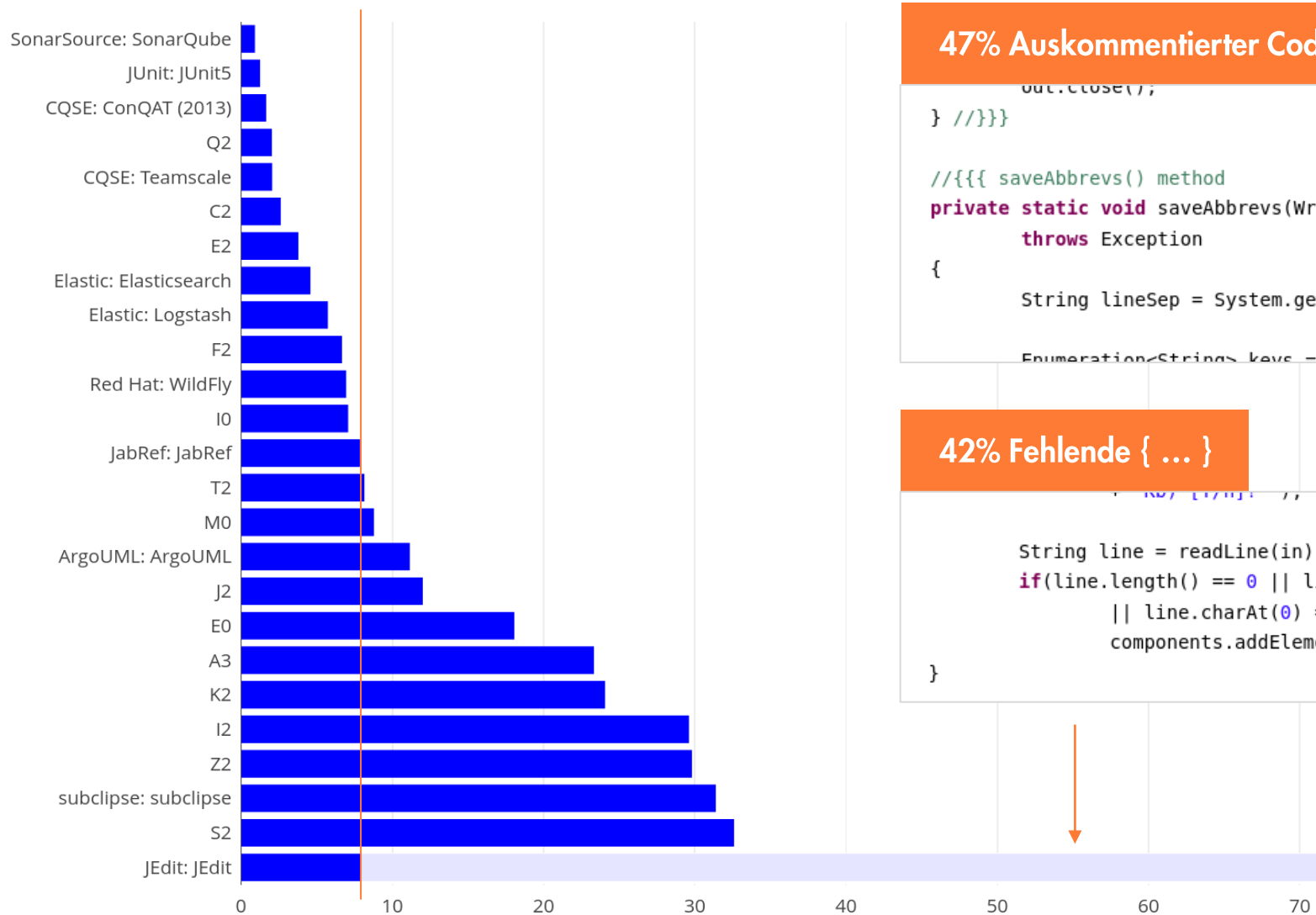




# Kommentierung



# Codeanomalien / 1.000 SLOC



47% Auskommentierter Code

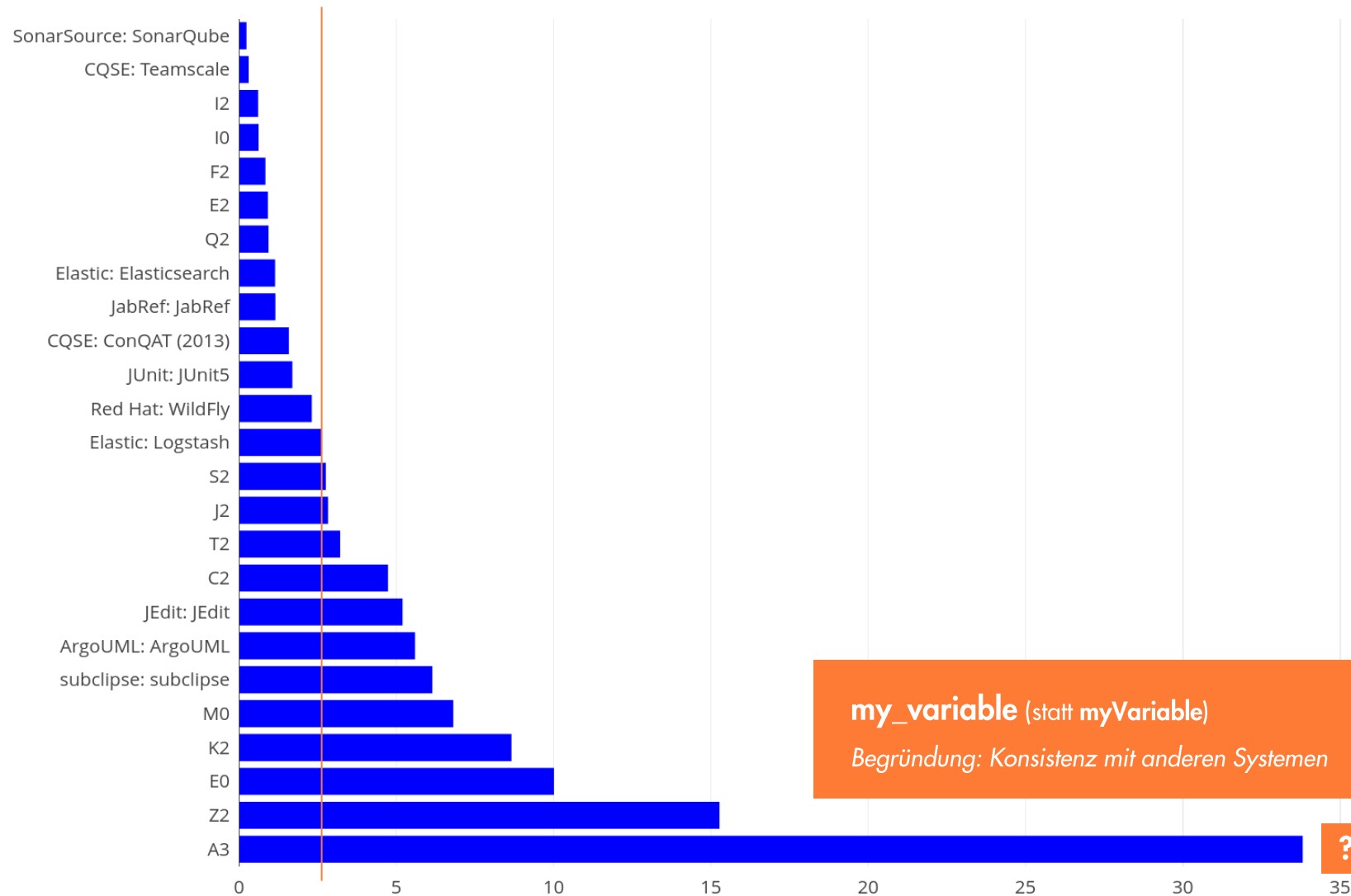
```
out.close();  
} //}}}  
  
//{{{ saveAbbrevs() method  
private static void saveAbbrevs(Writer out, Hashtable<String,Str  
    throws Exception  
{  
    String lineSep = System.getProperty("line.separator");  
  
    Enumeration<String> keys = abbrevs.keys();
```

42% Fehlende { ... }

```
String line = readLine(in);  
if(line.length() == 0 || line.charAt(0) == 'y'  
    || line.charAt(0) == 'Y')  
    components.addElement(fileset);  
}
```

?

# Verletzungen der Namenskonventionen / 1.000 SLOC





## Ein Benchmark ...



... ist ein wichtiger Baustein um von der **Messung zur Bewertung** zu kommen



... erlaubt von konkreten **Thresholds** und Check-Auswahl zu **abstrahieren**



... ist **keine automatische Bewertung** und muss interpretiert werden



... muss **gewissenhaft kuratiert** werden

**Handlungsbedarf ?**



Dr. Nils Göde · [goede@cqse.eu](mailto:goede@cqse.eu) · +49 176 10452662

**CQSE** GmbH  
Centa-Hafenbrädl-Straße 59  
81249 München



[cqse.eu/jax2021](https://cqse.eu/jax2021)

**CQSE**  
Continuous Quality in Software Engineering